(19) BUNDESREPUBLIK DEUTSCHLAND

DEUTSCHES PATENT- UND MARKENAMT

(12) **Offenlegungsschrift**

(10) **DE 199 26 538 A 1**

(51) Int. Cl.[7]:
G 06 F 15/80
G 06 F 9/38
H 03 K 19/173

(21) Aktenzeichen: 199 26 538.0
(22) Anmeldetag: 10. 6. 1999
(43) Offenlegungstag: 14. 12. 2000

(71) Anmelder:
Pact Informationstechnologie GmbH, 80807 München, DE

(74) Vertreter:
Pietruk, C., Dipl.-Phys., Pat.-Anw., 76229 Karlsruhe

(72) Erfinder:
Vorbach, Martin, 80689 München, DE

Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen

(54) Hardware und Betriebsverfahren

(57) Die Erfindung betrifft eine ein- oder mehrdimensionale Zellstruktur. Hierbei ist eine zeitlich getrennte Sequenzierung vorgesehen.

DE 199 26 538 A 1

DE 199 26 538 A 1

schen Einheiten, deren Verschaltung ebenfalls programmierbar und während des Betriebes umprogrammierbar ist. Derartige logische Bausteine sind unter dem Oberbegriff FPGA von verschiedenen Firmen verfügbar. Weiterhin sind mehrere Patente veröffentlicht, die spezielle arithmetische Bausteine mit automatischer Datensynchronisation und verbesserten offenlegen.

Sämtliche beschriebene Bausteine besitzen eine zwei- oder mehrdimensionale Anordnung von logischen und/oder arithmetischen Einheiten, die über Bussysteme miteinander verschaltbar sind.

Aufgabe der Erfindung ist es, ein Programmierverfahren zur Verfügung zu stellen, das es ermöglicht die beschriebenen Bausteine in gewöhnlichen Hochsprachen effizient zu programmieren und dabei die Vorteile der durch die Vielzahl von Einheiten entstehende Parallelität der beschriebenen Bausteine weitgehend automatisch, vollständig und effizient zu nutzen.

Stand der Technik

Bausteine der genannten Gattung werden zumeist unter Verwendung gewöhnlicher Datenflußsprachen programmiert. Dabei treten zwei grundlegende Probleme auf:

1. Die Programmierung in Datenflußsprachen ist für Programmierer gewöhnungsbedürftig, tief sequentielle Aufgaben lassen sich nur sehr umständlich beschreiben.
2. Große Applikationen und sequentielle Beschreibungen lassen sich mit den bestehenden Übersetzungsprogrammen (Synthese-Tools) nur bedingt auf die gewünschte Zieltechnologie abbilden (synthetisieren).

Für gewöhnlich werden Applikationen in mehrere Teilapplikationen partitioniert, die dann einzeln auf die Zieltechnologie synthetisiert werden (Fig. 1). Die einzelnen Binärcodes werden dann auf jeweils einen Baustein geladen. Wesentliche Voraussetzung der Erfindung ist das in DE 44 16 881 beschriebene Verfahren, das es ermöglicht, mehrere partitionierte Teilapplikationen innerhalb eines Bausteines zu nutzen, indem die zeitliche Abhängigkeit analysiert wird und über Steuersignale sequentiell die jeweils erforderlichen Teilapplikationen bei einer übergeordneten Ladeeinheit angefordert und von dieser daraufhin auf den Baustein geladen werden.

Existierende Synthese-Tools sind nur bedingt in der Lage Programm-Schleifen auf Bausteine abzubilden (Fig. 2 (0201)).

Dabei werden sogenannte FOR-Schleifen (0202) als Primitiv-Schleife häufig noch dadurch unterstützt, daß die Schleife vollkommen auf die Ressourcen des Zielbausteines ausgewalzt werden.

WHILE-Schleifen (0203) besitzen im Gegensatz zu FOR-Schleifen keinen konstanten Abbruchswert. Vielmehr wird durch eine Bedingung evaluiert, wann der Schleifenabbruch stattfindet. Daher ist gewöhnlicherweise (wenn die Bedingung nicht konstant ist) zur Synthesezeit nicht bekannt, wenn die Schleife abbricht. Durch das dynamische Verhalten können Synthese-Tools diese Schleifen nicht fest auf Hardware abgebildet d. h. auf einen Zielbaustein übertragen werden.

Rekursionen sind grundsätzlich nicht auf Hardware abbildbar, wann die Rekursionstiefe nicht zur Synthesezeit bekannt und damit konstant ist. Bei der Rekursion werden mit jeder neuen Rekursionsebene neue Ressourcen allokiert. Das würde bedeuten, daß mit jeder Rekursionsebene neue Hardware zur Verfügung gestellt werden muß, was aber dynamisch nicht möglich ist.

Selbst einfache Grundstrukturen sind von Synthesetools nur dann abbildbar, wenn der Zielbaustein ausreichend groß ist, d. h. ausreichende Ressourcen bietet.

Einfache zeitliche Abhängigkeiten (0301) werden durch heutige Synthese-Tools nicht in mehrere Teilapplikationen partitioniert und sind deshalb nur als Ganzes auf einen Zielbaustein übertragbar.

Bedingte Ausführungen (0302) und Schleifen über Bedingungen (0303) sind ebenfalls nur abbildbar, wenn ausreichende Ressourcen auf dem Zielbaustein existieren.

Erfindungsgemäßes Verfahren

Durch das in DE 44 16 881 beschriebene Verfahren ist es möglich Bedingungen zur Laufzeit innerhalb der Hardwarestrukturen der genannten Bausteine zu erkennen und derart dynamisch darauf zu reagieren, daß die Funktion der Hardware entsprechend der eingetretenen Bedingung modifiziert wird, was im wesentlichen durch das Konfigurieren einer neuen Struktur geschieht.

Ein wesentlicher Schritt in dem erfindungsgemäßen Verfahren ist die Partitionierung von Graphen in zeitlich unabhängige Teilgraphen.

Der Begriff "zeitliche Unabhängigkeit" wird damit definiert, daß die Daten, die zwischen zwei Teilapplikationen übertragen werden durch einen Speicher, gleich welcher Ausgestaltung (also auch mittels einfacher Register), entkoppelt werden. Die ist besonders an den Stellen eines Graphen möglich, an denen eine klare Schnittstelle mit einer begrenzten und möglichst minimalen Menge von Signalen zwischen den beiden Teilapplikationen besteht.

Die zeitliche Unabhängigkeit kann in großen Graphen durch das gezielte Einfügen von klar definierten und möglichste einfachen Schnittstellen zum Speichern von Daten in einen Zwischenspeicher herbeigeführt werden (vgl. S_n in Fig. 4). Schleifen weisen grundsätzlich eine starke zeitliche Unabhängigkeit auf, da sie lange Zeit über einer bestimmten Menge

von (zumeist) in der Schleife lokalen Variablen arbeiten und nur beim Schleifeneintritt und beim verlassen der Schleife eine Übertragung der Operanden bzw. des Ergebnisses erfordern.

Durch die zeitliche Unabhängigkeit wird erreicht, daß nach der vollständigen Ausführung einer Teilapplikation die nachfolgende Teilapplikation geladen werden kann, ohne daß irgendwelche weiteren Abhängigkeiten oder Einflüsse auftreten. Beim Speichern der Daten in den genannten Speicher kann ein Signal (Trigger) generiert werden, das die übergeordneten Ladeeinheit zum Nachladen der nächsten Teilapplikation auffordert. Der Trigger kann bei der Verwendung von einfachen Registern als Speicher immer generiert werden, wenn das Register beschrieben wird. Bei der Verwendung von Speichern, i. b. von solchen die nach dem FIFO-Prinzip arbeiten, ist die Generierung des Triggers von mehreren Bedingungen abhängig.

Folgende Bedingungen können beispielsweise einzeln oder kombiniert ein Trigger erzeugen:

- Ergebnis-Speicher voll
- Operanden-Speicher leer
- keine neuen Operanden
- Beliebige Bedingung innerhalb der Teilapplikation, generiert durch z. B.
- Vergleicher
- Zähler.

Eine Teilapplikation wird im folgenden auch Modul genannt, um die Verständlichkeit aus Sicht der klassischen Programmierung zu erhöhen. Aus demselben Grund werden Signale im folgenden auch Variablen genannt. Dabei unterscheiden sich die Variablen in einem Punkt wesentlich von herkömmlichen Variablen: Jeder Variable ist ein Statussignal (Ready) zugeordnet, das anzeigt, ob die Variable einen gültigen Wert besitzt. Wenn ein Signal einen gültigen (berechneten) Wert besitzt, ist das Statussignal Ready; wenn das Signal keinen gültigen Wert besitzt (Berechnung noch nicht abgeschlossen), ist das Statussignal Not_Ready. Das Prinzip ist ausführlich in der Patentanmeldung PACT02 beschrieben.

Das Prozessormodell

Die in den folgenden Figuren gezeigten Graphen besitzen als Graphenknoten immer in Modul, wobei davon ausgegangen wird, daß mehrere Module auf einen Zielbaustein abgebildet werden können. Das heißt, obwohl alle Module zeitlich voneinander unabhängig sind, wird nur bei nach den Modulen eine Umkonfiguration durchgeführt, bzw. ein Datenspeicher eingefügt, die mit einem vertikalen Strich und Δt markiert sind. Dieser Punkt wird Umkonfigurationszeitpunkt genannt.

Das bedeutet zusammenfassend:

1. Große Module können an geeigneten Stellen partitioniert werden und in kleine zeitlich voneinander unabhängige Module zerlegt werden.
2. Bei kleinen Modulen die sich gemeinsam auf einen Zielbaustein abgebildet werden können, wird auf die zeitliche Unabhängigkeit verzichtet. Dadurch werden Konfigurationsschritte eingespart und die Datenverarbeitung beschleunigt.
3. Die Umkonfigurationszeitpunkte werden entsprechend der Ressourcen der Zielbausteine positioniert. Dadurch ist eine beliebige Skalierung der Graphenlänge gegeben.

In Fig. 4a sind einige grundlegenden Eigenschaften des erfindungsgemäßen Verfahrens dargestellt:
Die Module des Types A sind zu einer Gruppe zusammengefaßt und besitzen am Ende einen bedingten Sprung, entweder nach B1 oder B2. An dieser Position (0401) ist ein Umkonfigurationspunkt eingefügt, da es sinnvoll ist die Zweige des bedingten Sprunges als jeweils eine Gruppe zu betrachten (Fall 1). Würden dagegen beide Zweige von B (B1 und B2) zusätzlich zu A auf den Zielbaustein passen (Fall 2), wäre es sinnvoll nur einen Umkonfigurationspunkt bei 0402 einzufügen, da dadurch die Zahl der Konfigurationen verringert wird und sich die Verarbeitungsgeschwindigkeit erhöht. Beide Zweige (B1 und B2) springen bei 0402 nach C.

Die Konfiguration der Zellen auf dem Zielbaustein ist in Fig. 4b schematisch dargestellt. Dabei werden die Funktionen der einzelnen Graphenknoten auf die Zellen des Zielbausteins abgebildet. Jeweils eine Zeile stellt eine Konfiguration dar. Die gestrichelten Pfeile bei einem Zeilenwechsel zeigen eine Umkonfiguration. S_n ist eine datenspeichernde Zelle, von beliebiger Ausgestaltung (Register, Speicher, etc.). Dabei ist S_nI ein Speicher, der Daten entgegennimmt und S_nO ein Speicher der Daten ausgibt. Der Speicher S_n ist für gleiche n jeweils derselbe, I und O kennzeichnen die Datentransferrichtung.

Beide Fälle des bedingten Sprunges (Fall 1, Fall 2) sind dargestellt.

Das Modell in Fig. 4 entspricht einem Datenflußmodell, jedoch mit der wesentlichen Erweiterung der Umkonfigurationspunkt und der damit erreichbaren Partitionierung des Graphen, wobei die zwischen den Partitionen übertragenen Daten zwischengespeichert werden.

Im Modell von Fig. 5a wird aus einer beliebigen Graphenmenge und -Konstellation (0501) selektiv ein Graph aus einer Menge von Graphen B aufgerufen. Nach der Ausführung von B gelangen die Daten nach 0501 zurück.

Wird in 0501 ein ausreichend großer Sequencer (A) implementiert, ist mit dem Modell ein den typischen Prozessoren sehr ähnliches Prinzip implementierbar. Dabei gelangen

1. Daten in den Sequenzer A, die dieser als Befehle dekodiert und entsprechend dem "von Neumann"-Prinzip darauf reagiert;
2. Daten in den Sequenzer A, die als Daten betrachtet werden und an ein fest konfiguriertes Rechenwerk C zur Berechnung weitergeleitet werden.

Fig. 5b schematisiert die Abbildung auf die einzelnen Zellen, wobei in 0502 der pipelineartige Rechenwerks-Character symbolisiert wird.

Während in den Umkonfigurationspunkten von Fig. 4 vorzugsweise größere Speicher zum Zwischenspeichern der Daten eingefügt werden, ist eine einfache Synchronisation der Daten in den Umkonfigurationspunkten von Fig. 5 ausreichend, da der Datenstrom vorzugsweise als ganzer durch den Graphen B läuft und der Graph B nicht weiter partitioniert ist; dadurch ist das Zwischenspeichern der Daten überflüssig.

In Fig. 6a sind verschiedene Schleifen dargestellt. Schleifen können grundsätzlich auf drei Arten behandelt werden:

1. Hardware-Ansatz: Schleifen werden vollständig ausgewalzt auf die Zielhardware abgebildet (0601a/b). Wie bereits erläutert ist dies nur bei wenigen Schleifenarten möglich.
2. Datenfluß-Ansatz: Innerhalb der Datenflusses werden Schleifen über mehrere Zellen hinweg aufgebaut (0602a/b). Das Ende der Schleife wird auf den Schleifenanfang rückgekoppelt.
3. Sequenzer-Ansatz: Ein Sequenzer mit minimalem Befehlssatz führt die Schleife aus (0603a/b). Dabei sind die Zellen der Zielbausteine so ausgestaltet, daß sie den entsprechenden Sequenzer beinhalten (vgl. Fig. 11a/b).

Durch eine geeignete Zerlegung von Schleifen kann deren Ausführung ggf. optimiert werden:

1. Unter Verwendung von Optimierungsmethoden nach dem Stand der Technik läßt sich häufig der Schleifenrumpf, also der wiederholt auszuführende Teil, dadurch optimieren, daß bestimmte Operationen aus der Schleife entfernt werden und vor oder hinter die Schleife gestellt werden (0604a/b). Dadurch wird die Menge der zu sequencenden Befehle erheblich reduziert. Die entfernten Operationen werden nur einmal vor bzw. nach Ausführung der Schleife durchlaufen.
2. Eine weitere Optimierungsmöglichkeit ist das Teilen von Schleifen in mehrere kleinere oder kürzere Schleifen. Dabei findet die Teilung derart statt, daß mehrere parallele oder mehrere sequentielle (0605a/b) Schleifen entstehen.

Fig. 7 verdeutlicht die Implementierung einer Rekursion. Dabei werden dieselben Ressourcen (0701) in Form von Zellen für jede Rekursionsebene (1–3) verwendet. Die Ergebnisse einer jeden Rekursionsebene (1–3) werden beim Aufbau (0711) in einen nach dem Stack-Prinzip aufgebauten Speicher (0702) geschrieben. Gleichzeitig mit dem Abbau (0712) der Ebenen wird der Stack abgebaut.

Hochsprachenbeispiele

Ein Modul kann beispielsweise folgendermaßen deklariert werden:

```
module example
input  (var1, var2 : ty1; var3 : ty2).
output (res1, res2 : ty3).
begin
...
register <regname1> (res1).
register <regname2> (res2).
terminate@ (res1 & res2; 1).
end.
```

module kennzeichnet den Beginn eines Modules.
input/output definiert die Ein-/Ausgangsvariablen mit den Typen ty_n.
begin . . . end markieren den Rumpf des Modules.
register <regname1/2> übergibt das Ergebnis an den Output, wobei des Ergebnis in dem durch <regname1/2> spezifizierten Register zwischengespeichert wird. <regname1/2> ist dabei eine globale Referenz auf ein bestimmtes Register.

Als weitere Übergabemodi an den Output stehen beispielsweise folgende Speicherarten zur Verfügung:
fifo <fifoname>, wobei die Daten an einen nach dem FIFO-Prinzip arbeitenden Speicher übergeben werden. fifoname ißt dabei eine globale Referenz auf einen bestimmten, im FIFO-Modus arbeitenden Speicher. terminate@ wird dabei um den Parameter bzw. das Signal "fifofull" erweitert, der/das anzeigt, daß der Speicher voll ist.
stack <stackname>, wobei die Daten an einen nach dem Stack-Prinzip arbeitenden Speicher übergeben werden. stack-

ten Ladeeinheit das nachfolgende Modul lädt.

```
module example2
input  (var1, var2 : ty3; var3 : ty2).
output (res1 : ty4).
begin
register <regname1> (var1, var2).
...
fifo <fifoname1> (res1, 256).
terminate@ (fifofull(<fifoname1>); 1).
end.
```

register wird in diesem Beispiel über input-Daten definiert. Dabei ist <regname1> derselbe wie in example1. Dies bewirkt, daß das Register, das die output-Daten in example1 aufnimmt, die input-Daten für example2 zur Verfügung stellt. fifo definiert einen FIFO-Speicher der Tiefe 256 für die Ausgangsdaten res1. Das Full-Flag (fifofull) des FIFO-Speichers wird in terminate@ als Abbruchkriterium verwendet.

```
module main
input  (in1, in2 : ty1; in3 : ty2).
output (out1 : ty4).
begin
define <regname1> : register(234).
define <regname2> : register(26).
define <fifoname1> : fifo(4).
...
(var12, var72) = call example1 (in1, in2, in3).
(out1) = call example2 (var12, var72, var243).
...
signal (out1).
terminate@ (example2).
end.
```

define definiert eine Schnittstelle für Daten (Register, Speicher, etc.). Bei der Definition werden die erforderlichen Ressourcen, sowie die Bezeichnung der Schnittstelle angegeben. Da die Ressourcen eindeutig angegeben werden und nur einmal verwendet werden können, ist die Definition global, d. h. die Bezeichnung gilt für das gesamte Programm.
call ruft ein Modul als Unterprogramm auf.
signal definiert ein Signal als Ausgangssignal, ohne daß eine Zwischenspeicherung verwendet wird.

Durch terminate@ (example2) wird das Modul main terminiert, sobald das Unterprogramm example2 terminiert.

Durch die globale Deklaration "define ..." ist es prinzipiell nicht mehr notwendig, die so definierten input/output Signale in die Schnittstellen-Deklaration der Module aufzunehmen. Die entsprechend modifizierten Beispiel-Module würden dann folgendermaßen aussehen:

Zur Bestimmung der Zustände innerhalb eines Graphen werden die Statusregister der einzelnen Zellen (PAEs) über ein zusätzlich zum Datenbus (0801) existierendes Status-Bussystem (0802) allen anderen Rechenwerken zur Verfügung gestellt (Fig. 8b). Das bedeutet, daß eine Zelle (PAE X) die Statusinformation einer anderen Zelle (PAE Y) evaluieren kann und dementsprechend die Daten verarbeitet. Um den Unterschied zu bestehenden Parallerechnersystemen zu verdeutlichen, ist in Fig. 8a der Stand der Technik angegeben. Dabei ist ein Multiprozessorsystem gezeigt, dessen Prozessoren über einen gemeinsamen Datenbus (0803) miteinander verbünden sind. Ein explizites Bussystem für den synchronen Austausch von Daten und Status existiert nicht.

Abschließend soll angemerkt werden, daß je nach Aufgabe sowohl der Datenflußgraph, als auch der Kontrolliflußgraph entsprechend dem beschriebenen Verfahren behandelt werden kann.

Erweiterungen in der Hardware gegenüber PACT02 und PACT04

Durch PACT02 und PACT04 ist der Stand der Technik in Bezug auf die Konfigurationseigenschaften von Zellen (PAEs)definiert und veröffentlicht in DE 196 51 075 (PACT02) sowie in DE 196 54 846 (PACT04).

Dabei soll auf zwei Eigenschaften eingegangen werden:

1. Einer PAE ist gemäß PACT02 ein Satz von Konfigurationsregistern zugeordnet, der eine Konfiguration beinhaltet (Fig. 8a).
2. Eine Gruppe von PAEs kann gemäß PACT04 auf einen Speicher zum Speichern oder Lesen von Daten zugreifen (Fig. 8b).

Aufgabe ist es,

a) ein Verfahren zu schaffen, das das Umkonfigurieren von. PAEs beschleunigt und zeitlich von der übergeordneten Ladeeinheit entkoppelt, und

b) das Verfahren so auszulegen, daß gleichzeitig die Möglichkeit geschaffen wird über mehrere Konfigurationen zu Sequenzen.

Entkopplung der Konfigurationsregister

Das Konfigurationsregister wird von der übergeordneten Ladeeinheit (CT) entkoppelt (Fig. 9), indem ein Satz von mehreren Konfigurationsregistern (0901) verwendet wird. Immer genau eines der Konfigurationsregister bestimmt selektiv die Funktion dar PAE. Die Auswahl des aktiven Registers wird über einen Multiplexer (0902) durchgeführt. In jedes der Konfigurationsregister kann die CT beliebig schreiben, sofern dieses nicht die aktuelle Konfiguration der PAE bestimmt. Welches Konfigurationsregister von 0902 selektiert wird kann durch verschiedene Quellen bestimmt werden:

1. Ein beliebiges Status-Signal oder eine Gruppe beliebiger Status-Signale, die über ein Bussystem (0802) an 0902 geführt werden (Fig. 9a). Die Status-Signale werden dabei von beliebigen PAEs generiert oder durch externe Anschlüsse des Bausteins zur Verfügung gestellt (vgl. Fig. 8).
2. Das Status-Signal der PAE, die von 0901/0902 konfiguriert wird, dient zur Selektion (Fig. 9b).
3. Ein von der übergeordneten CT generiertes Signal dient zur Selektion (Fig. 9c).

Dabei ist es möglich wahlweise die eingehenden Signale (0903, 0904, 0905) mittels eines Registers für einen bestimmten Zeitraum zu speichern.

Durch den Einsatz mehrere Register wird die CT zeitlich entkoppelt. Das bedeutet, die CT kann mehrere Konfigurationen "vorladen", ohne daß eine direkte zeitliche Abhängigkeit besteht.

Lediglich das selektierte Register in 0901 noch nicht geladen ist, wird mit der Konfiguration der PAE so lange gewartet, bis die CT das Register geladen hat. Um festzustellen, ob ein Register eine gültige Information besitzt kann ein "Valid-Bit" (0906) pro Register eingeführt werden, das von der CT gesetzt wird. Ist 0906 bei einem selektierten Register nicht gesetzt, wird über ein Signal die CT zum schnellstmöglichen Setzten des Registers aufgefordert.

Das in Fig. 9 beschriebene Verfahren ist einfach zu einem Sequenzer erweiterbar (Fig. 10). Dazu wird ein Mikrokontroller (1001) zur Ansteuerung der Selektionssignale des Multiplexers (0902) verwendet. Der Sequenzer bestimmt dabei abhängig von der aktuell selektierten Konfiguration (1002) und einer zusätzlichen Statusinformation (1003/1004) die nächste zu selektierende Konfiguration. Dabei kann die Statusinformation

(a) der Status der Status-Signal der PAE, die von 0901/0902 konfiguriert wird sein (Fig. 10a).
(b) ein beliebiges über 0802 zugeführtes Statussignal sein (Fig. 10b).
(c) eine Kombination aus (a) und (b) sein.

Zum einfachen Verständnis kann 0901 als ein Speicher betrachtet werden, wobei über 0902 ein Befehl von 1001 adressiert wird. Die Adressierung ist dabei abhängig vom Befehl selbst und von einem Statusregister. Insoweit entspricht der

Wichtig ist, daß der Sequenzer dabei Sprünge, insbesondere auch bedingte Sprünge, innerhalb von 0901 ausführen kann.

Ein weiteres zusätzliches oder alternatives Verfahren (Fig. 11) zum Aufbau von Sequenzern innerhalb der genannten Bausteine ist die Verwendung der internen Datenspeicher (1101) zum Speichern der Konfigurationsinformation für eine PAE. Dabei wird der Datenausgang eines Speichers auf einen Konfigurationseingang einer PAE geschaltet (1102). Die Adresse (1103) für 1101 kann dabei von derselben PAE oder einer beliebigen anderen generiert werden.

Bei diesem Verfahren ist der Sequenzer nicht fest implementiert, sondern wird durch eine PAE oder eine Gruppe von PAEs nachgebildet.

Patentansprüche

1. Verfahren zum Ausführen von Programmen auf einem Baustein mit ein- oder mehrdimensionaler Zellstruktur, **dadurch gekennzeichnet**, dass Datenfluss- oder Kontrollflussgraphen in zeitlich getrennte Teilgraphen partitioniert werden und sequentiell auf den Baustein abgebildet und ausgeführt werden.
2. Hardware mit entkoppeltem Konfigurationsregister.

Hierzu 12 Seite(n) Zeichnungen

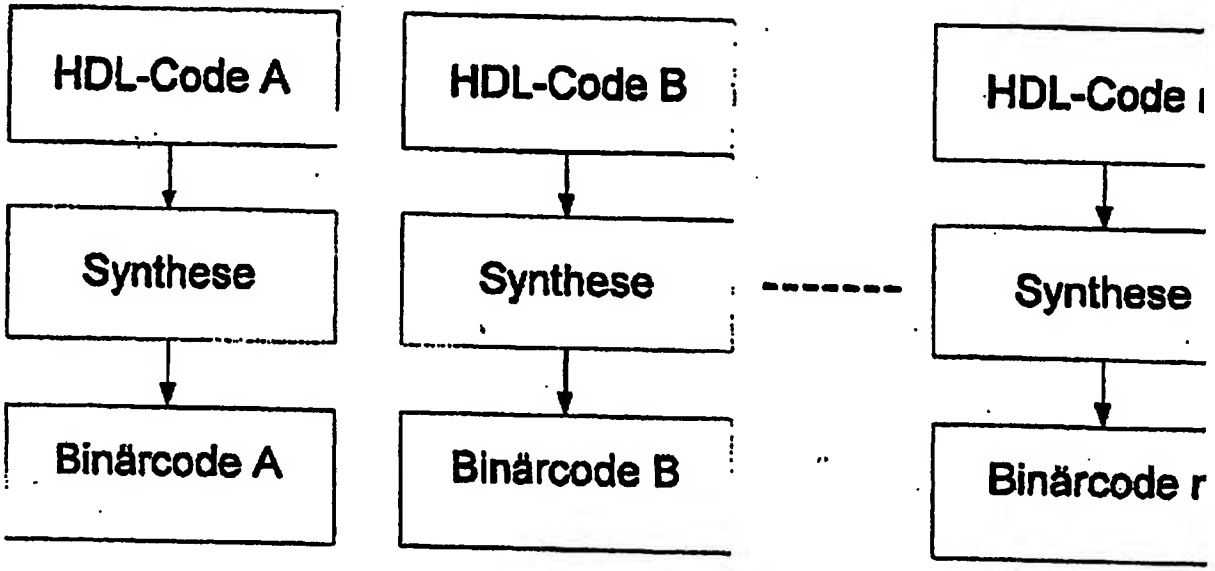


Fig. 1 **Stand der Technik**

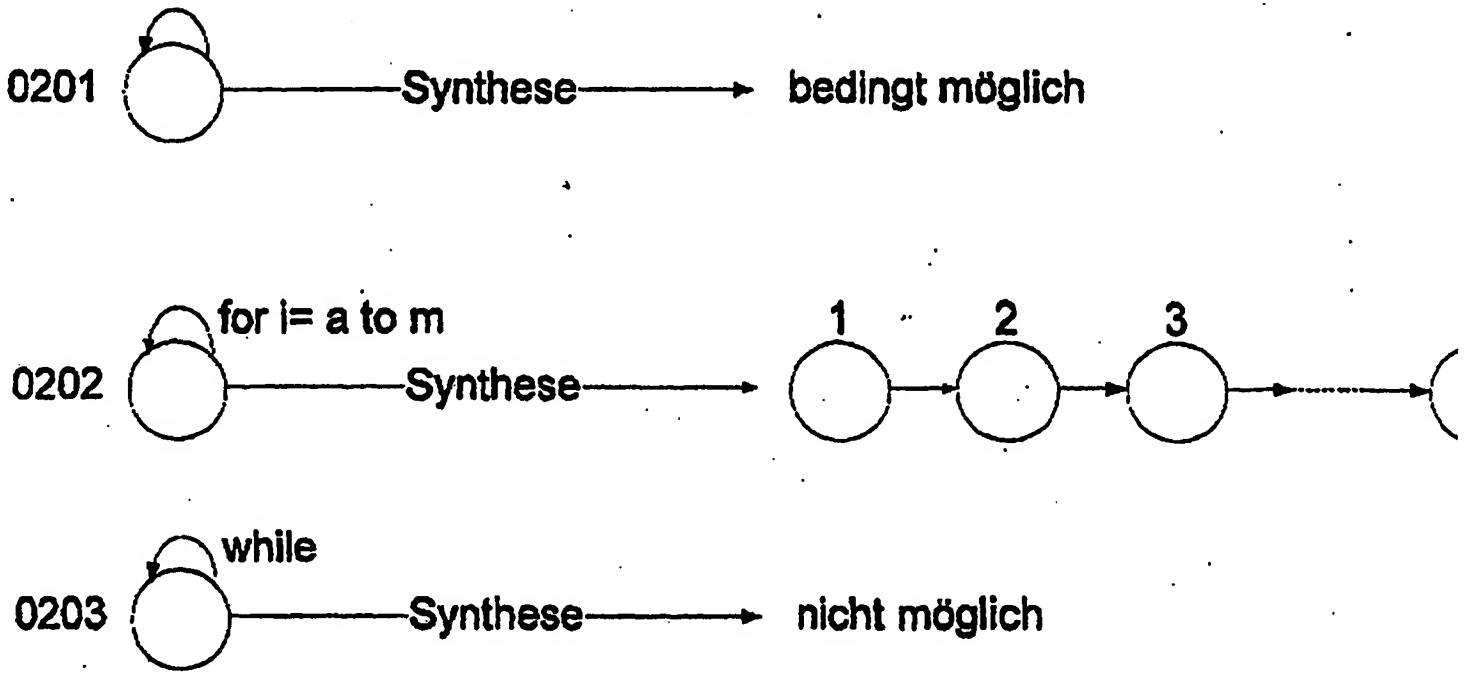


Fig. 2 Stand der Technik

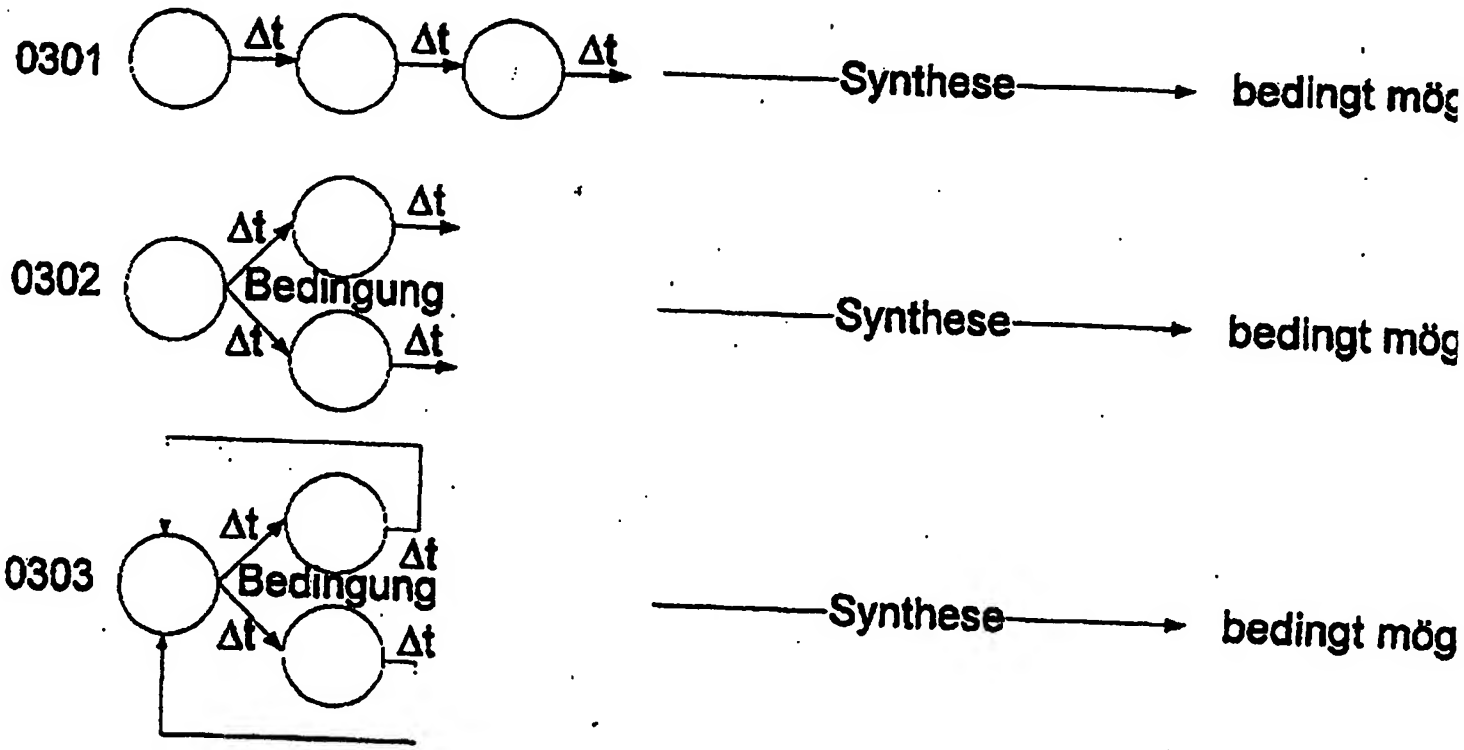


Fig. 3 Stand der Technik

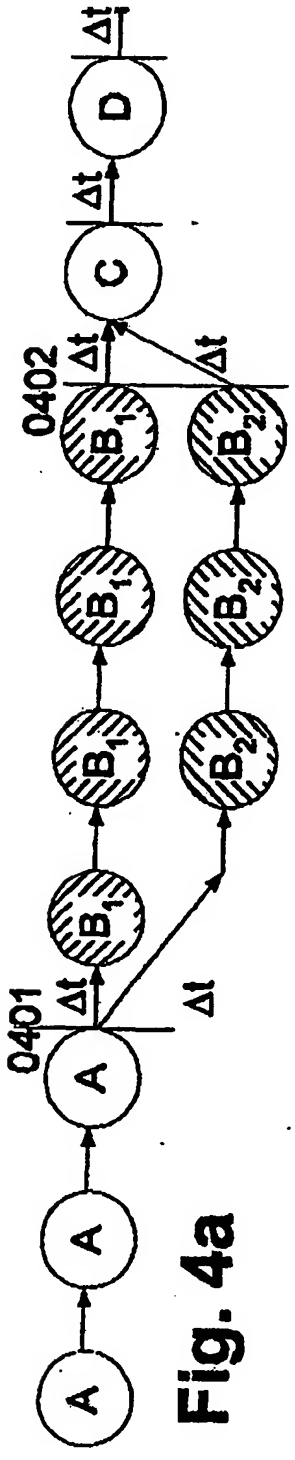

A
A
A
0401
Δt
Δt
B_1
B_1
B_2
B_1
B_2
0402
Δt
Δt
C
Δt
D
Δt

Fig. 4a

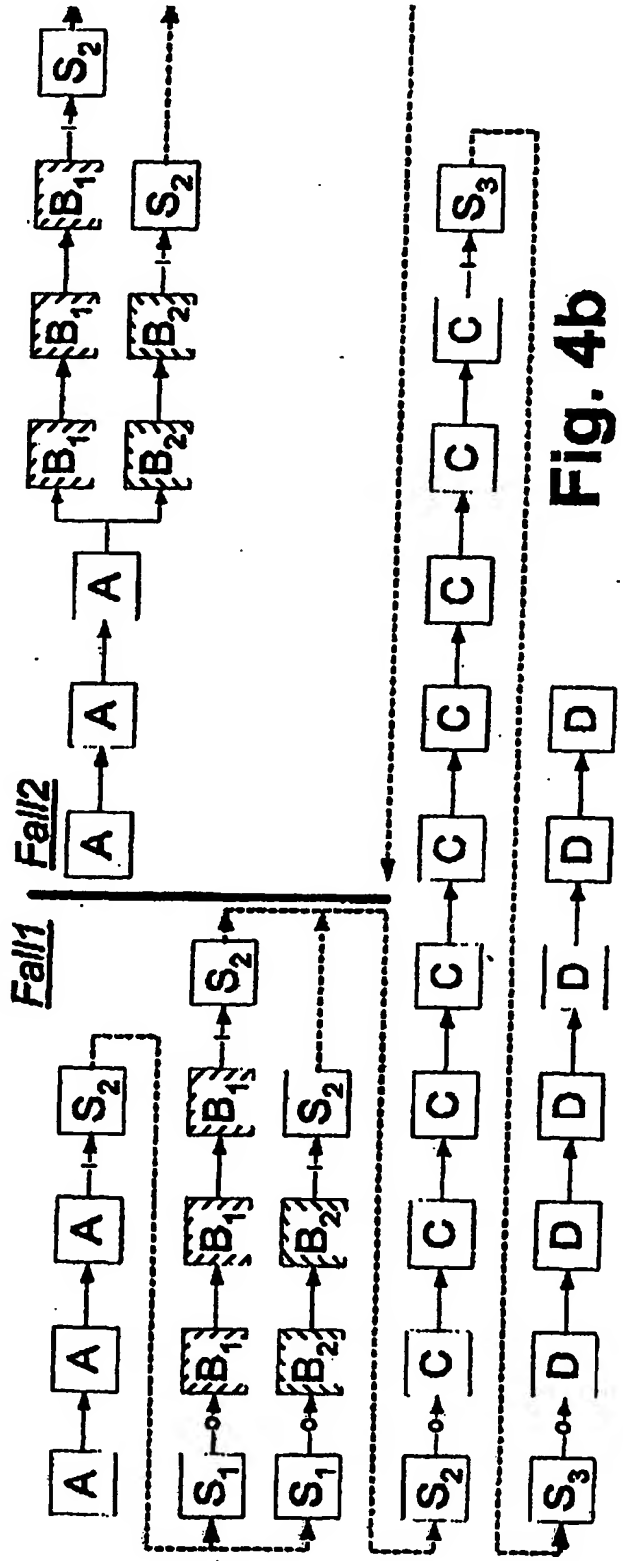

Fall1
Fall2
A
A
A
A
S_2
B_1
B_2
S_1
C
C
C
C
C
S_3
D
D
D
D

Fig. 4b

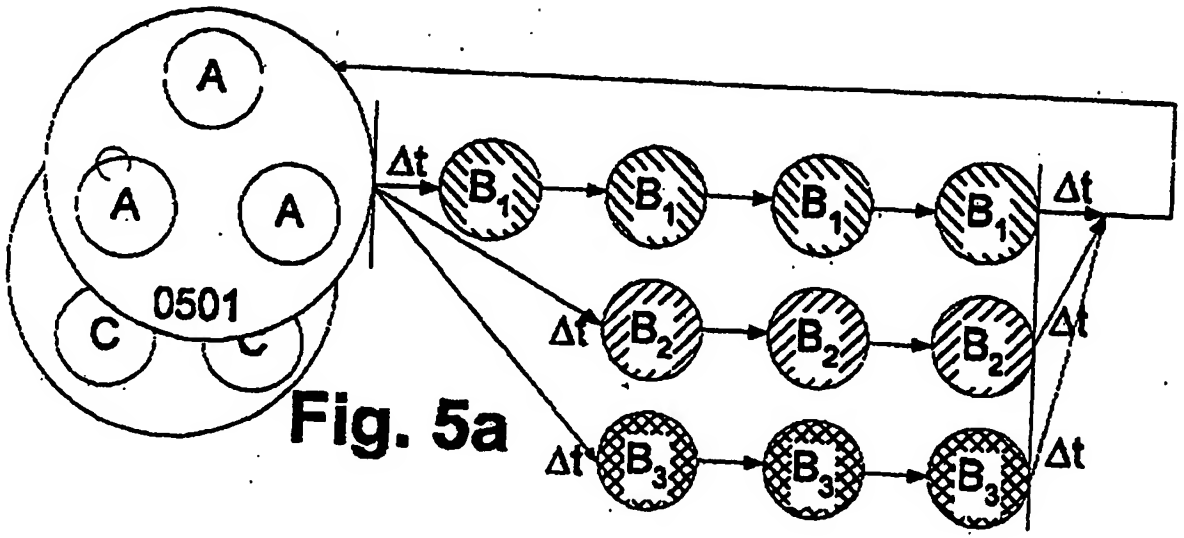


Fig. 5a

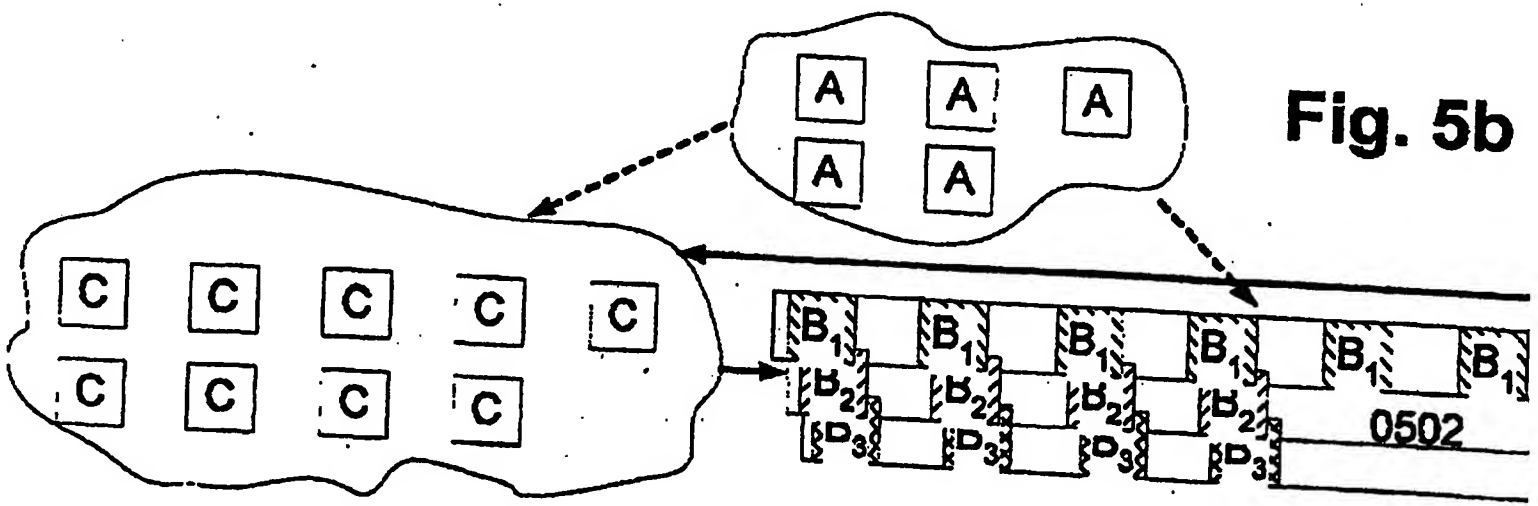


Fig. 5b

002 050481

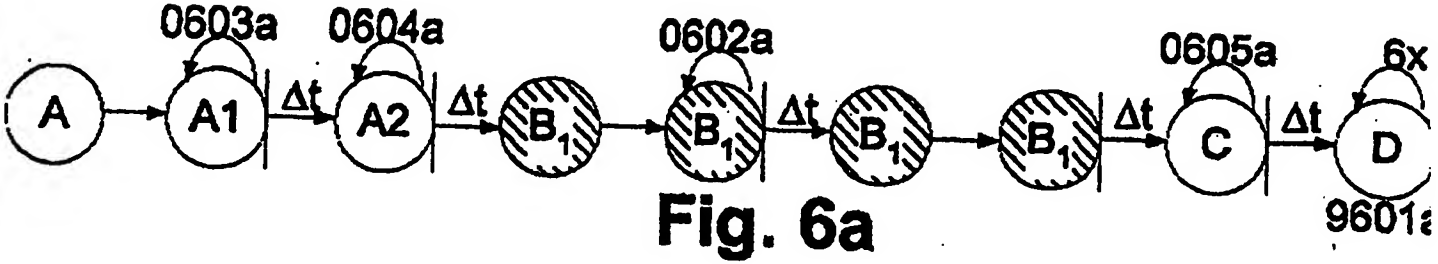


Fig. 6a

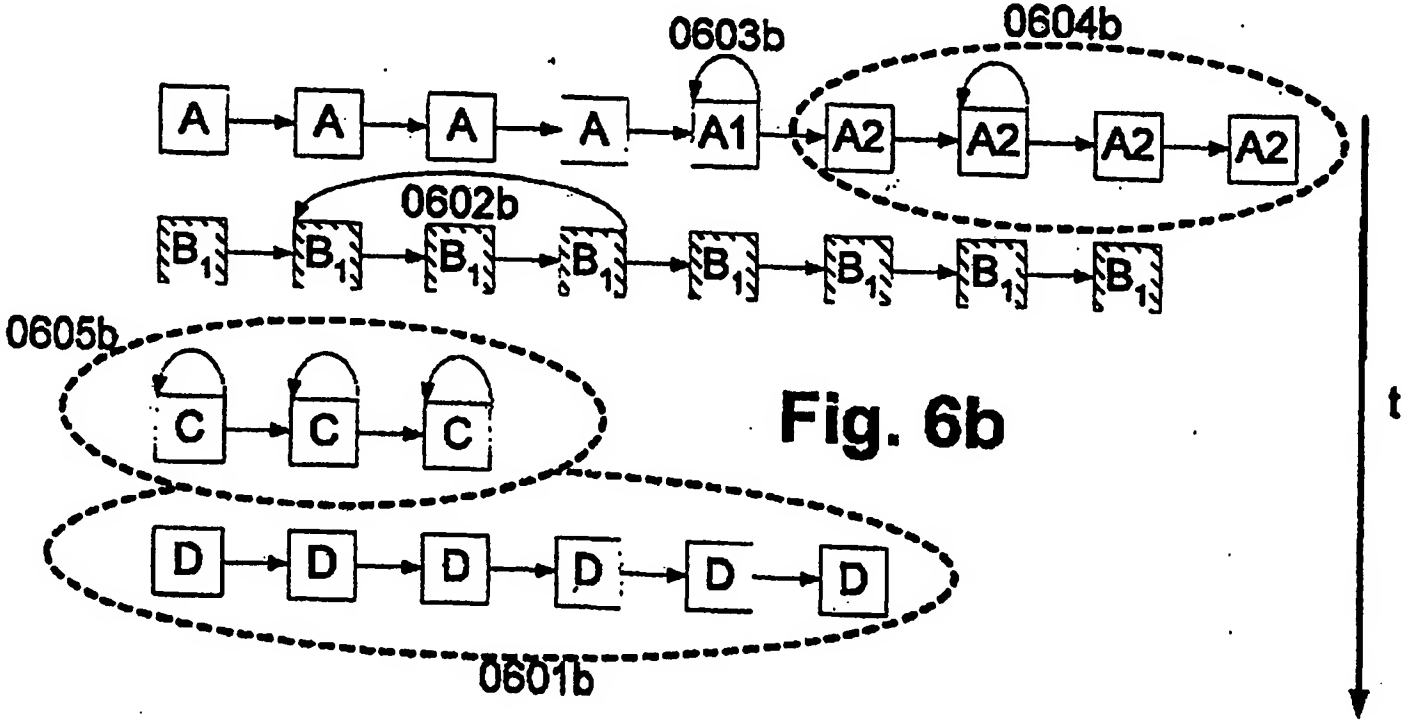


Fig. 6b

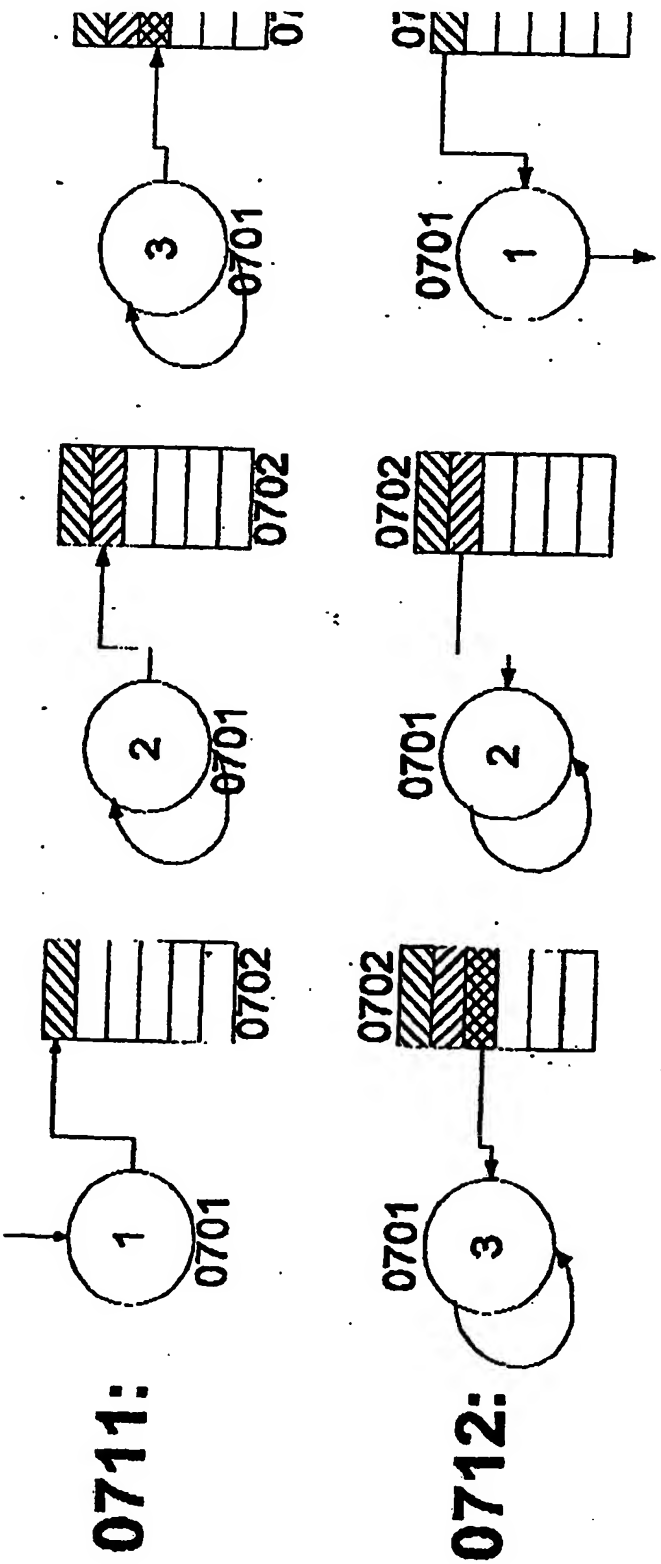


Fig. 7

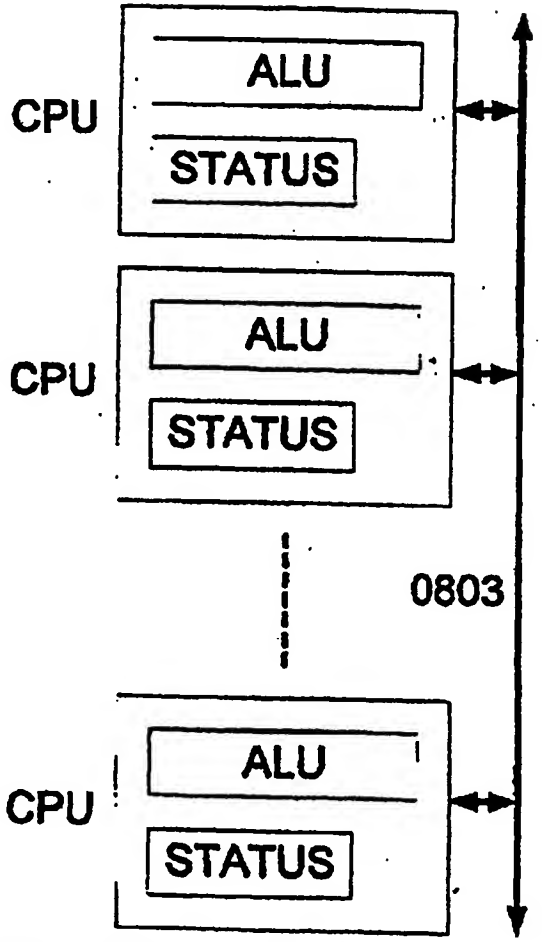


Fig. 8a Stand der Technik

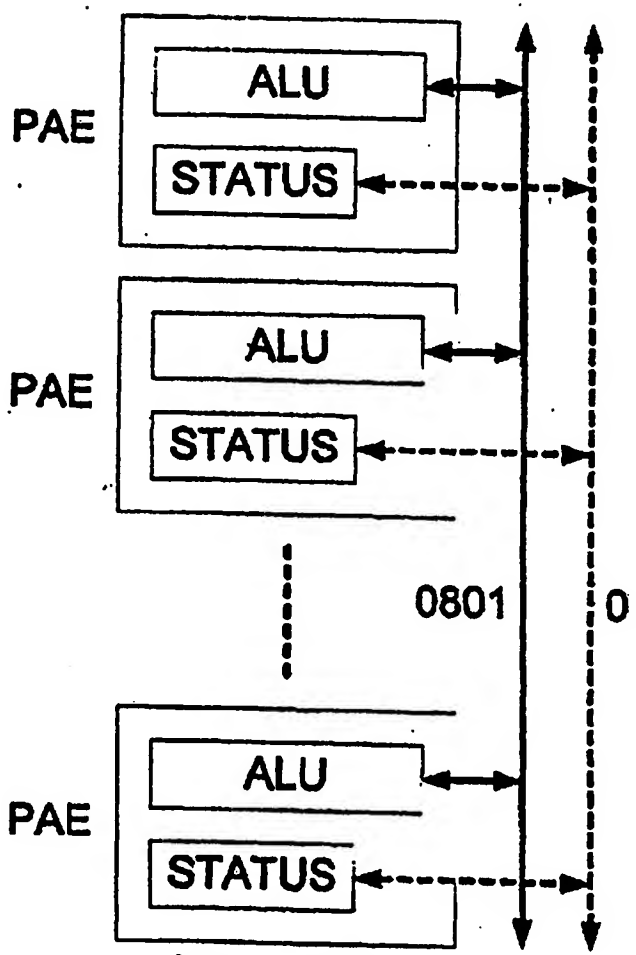


Fig. 8b

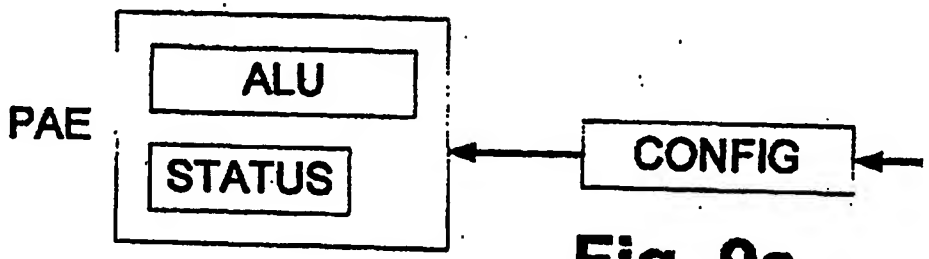
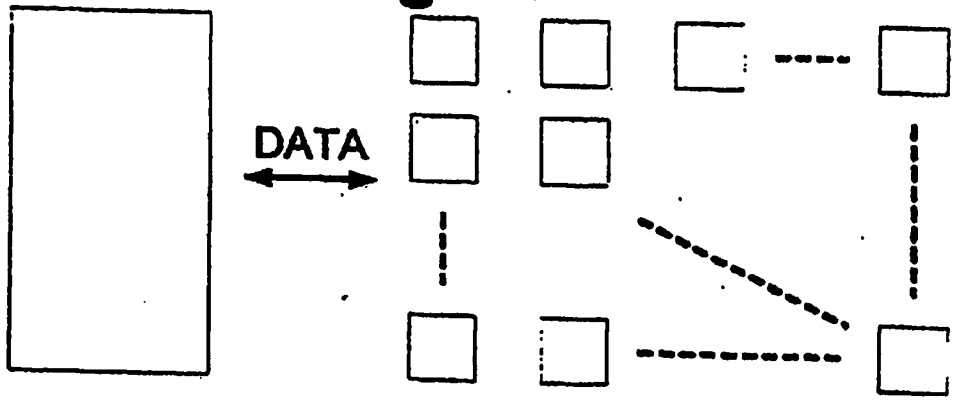


Fig. 9a

Stand der Technik

Fig. 9b



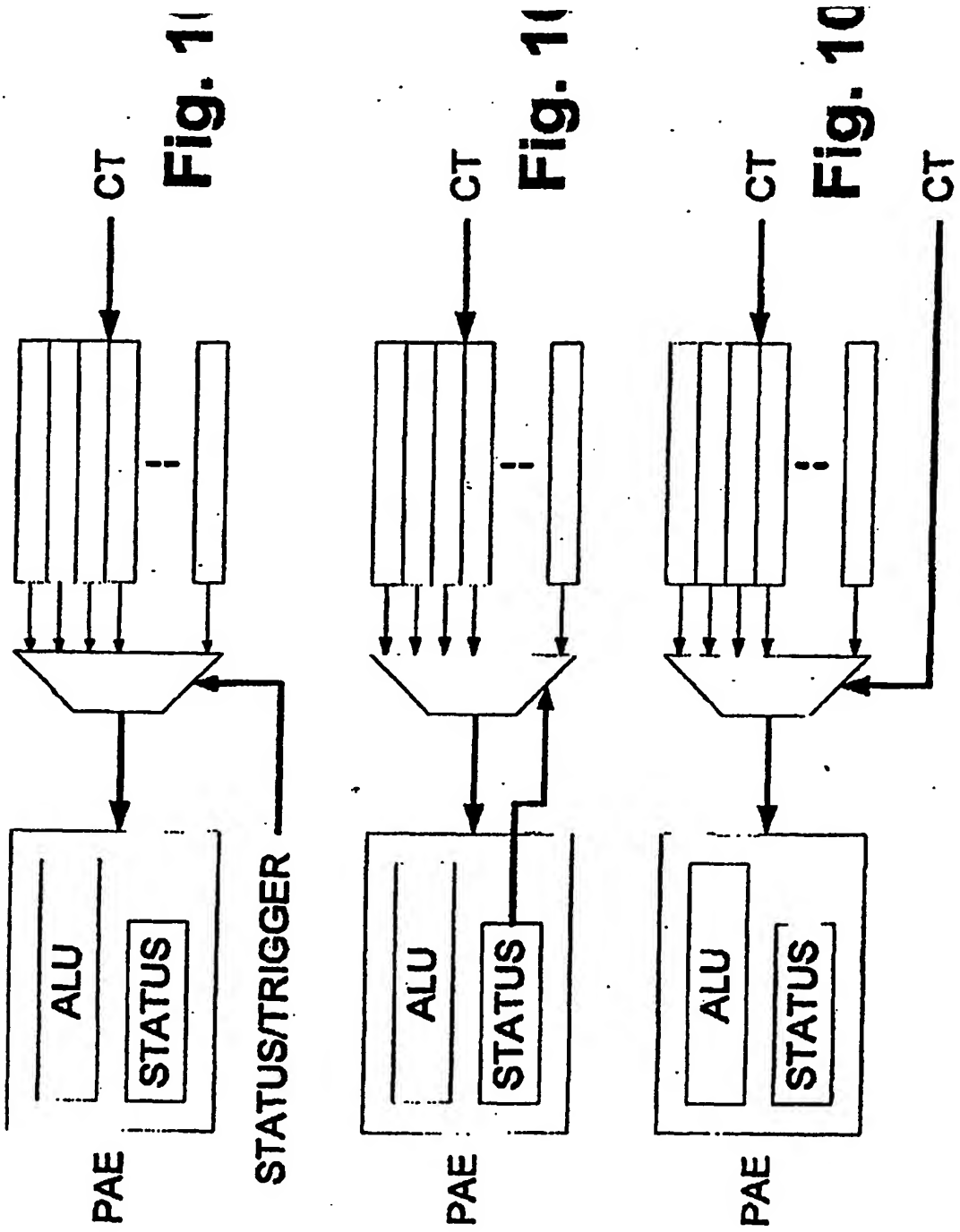


Fig. 1

Fig. 1

Fig. 1

Fig. 1

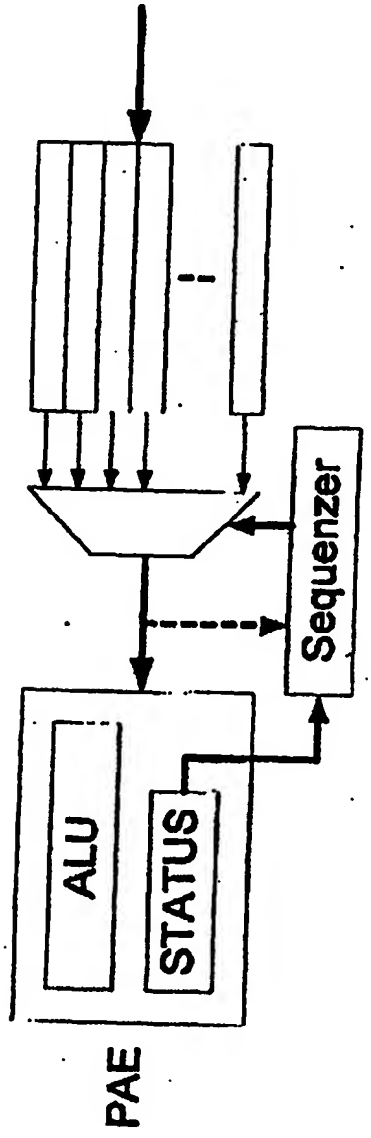

CT
Sequenzer
ALU
STATUS
PAE

Fig. 11

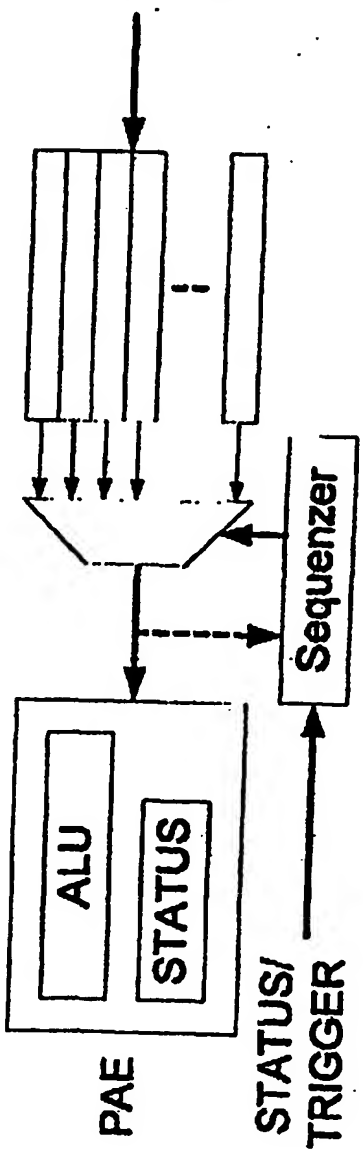

CT
Sequenzer
ALU
STATUS
PAE
STATUS/
TRIGGER

Fig. 12